SIGNAL PROCESSING ALGORITHMS
GEORGIA TECH ADA BENCHMARK

SPECIAL TECHNICAL REPORT

REPORT NO. STR–0142–91–014

AUGUST 5, 1991

GUIDANCE, NAVIGATION AND CONTROL
DIGITAL EMULATION TECHNOLOGY LABORATORY

Contract No. DASG60–89–C–0142

Sponsored By

The United States Army Strategic Defense Command

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332–0540

Contract Data Requirements List Item A008

Period Covered: Not Applicable

Type Report: As Required

DISCLAIMER

DISCLAIMER STATEMENT – The views, opinions, and / or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

DISTRIBUTION CONTROL

(1) DISTRIBUTION STATEMENT – Approved for public release; distribution is unlimited.

(2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227–7013, October 1988.

SIGNAL PROCESSING ALGORITHMS
GEORGIA TECH ADA BENCHMARK

AUGUST 5, 1991

RANI INDAHENG

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology
Atlanta, Georgia 30332–0540

Eugene L. Sanders
USASDC
Contract Monitor

Cecil O. Alford
Georgia Tech
Project Director

---

1 INTRODUCTION

This document describes a set of signal–processing algorithms, as implemented by the Computer Engineering Research Laboratory at Georgia Tech. The routines are presented as a representative collection of operations for processing Infrared Focal–Plane Array signals. This Ada version of the benchmark follows an earlier Fortran version (Special Technical Report STR–0142–90–008).

For the purposes of testing and dissemination, each algorithm is presented as a stand–alone Ada program. These programs are based upon a core *harness* routine which supports the input/output of a common data format (Georgia Tech Algorithm Evaluation Data Format – described in the Harness section). The modular implementations offer several benefits:

- simplification of the generation of test vectors for the verification of alternate implementations
- capability for testing various algorithm combinations, without re–compilation
- support for multiple language and/or processor–platform implementations

2 HARNESS

2.1 Description

The *harness* program shown below is the basis of the input/output methodology used by all of the routines in this document. The code implements a simple Pass–Through module which reads a data stream, picking off the FPA pixel data, and writing the data onto an output data stream.

The Georgia Tech Algorithm Evaluation Data Format is a simple ASCII text representation of a data stream. The data stream has two major components – the *Field Header* and the *Field Data*. The harness of each module processes the data stream by reading each line and checking for Field Headers which are relevant to that module. Any lines which are not relevant, or unrecognized, are immediately placed upon the output data stream. As soon as a relevant Field Header is recognized, the Field Data which follows is processed in a manner which is appropriate to that module and Field Header. This scheme provides for the chaining of modules output–to–input, without either module requiring knowledge of all, or any, of the other module's data formats. In typical use, controls for many modules could be included in a single data stream; each module would only process data intended for it. For example, suppose a test setup was composed of the following pipeline:

Input data stream ––> Spatial Filter ––> Simple Threshold ––> Output Data Stream

The data stream might appear as follows:

Input Data Stream	Description	Used by	Action
Dimensions 128	Field Header Field Data	Spatial Filter and Simple Threshold	input
Simple Thresholding Limits 0 256	Field Header Field Data	Simple Threshold	input
Pixel Data 99 93 ... 81 76	Field Header Field Data Field Data . . .	Spatial Filter and Simple Threshold	input, modified
End	Field Header	Spatial Filter and Simple Threshold	input

Output Data Stream	Description	Generated by	Action
Dimensions 128	Field Header Field Data	Spatial Filter and Simple Threshold	copied to output data stream
Simple Thresholding Limits 0 256	Field Header Field Data	Simple Threshold	copied to output data stream
Simple Thresholding Statistics 0 256 1024	Field Header Field Data	Simple Threshold	generated, then placed on output data stream
Pixel Data 99 93 ... 81 76	Field Header Field Data Field Data . . .	Spatial Filter and Simple Threshold	modified, then placed on output data stream
End	Field Header	Spatial Filter and Simple Threshold	copied to output data stream

2.2 Module Listing

```
--    Program      : HARNESS.ada
--                   This program acts as a harness for testing
--                   various Signal Processing Routines.
--    Original     : Occam Code -> Fortran Code (MicroSoft)
--    Author       : Rani Indaheng
--                   A. M. Henshaw
--    Date written: Jan 23, 1990
```

```
--    Note          : Conforms to the Ga. Tech Algorithm Evaluation data
--                    format
-----------------------------------------------------------------------
--    Contact       : Rani Indaheng        (404)894-3802
--                    Andrew Henshaw       (404)894-2521
--    Where         : Computer Engineering Research Laboratory
--                    Georgia Institute of Technology
--                    400 Tenth St. CRB 390
--                    Atlanta, GA 30332-0540
-----------------------------------------------------------------------
--    Update log :
--    When      Who             What
--    02.06.91  Rani Indaheng   Convert from Fortran to Ada
--    02.13.91  Rani Indaheng   Tested
-----------------------------------------------------------------------
with Text_io; use Text_io;
procedure HARNESS is
   package Int_io is new Integer_io(integer); use Int_io;
   type TwoDimType is array (integer range <>, integer range <>) of
        integer;
   MaxSize: constant integer := 64;
   subtype MaxSizeRange is positive range 1..MaxSize;
   subtype MaxStrSize is positive range 1..72;
   subtype StrType is string(MaxStrSize);
   -- Valid section headers
   Dim    : constant string := "Dimensions";
   Pixels: constant string := "Pixel Data";
   Done   : constant string := "End";

   RunFlag: boolean := TRUE;
   N, StrCount: integer;
   InData, OutData: TwoDimType(MaxSizeRange, MaxSizeRange);
   Header: StrType;

   procedure PASSTHRU (N: in integer;
                       InData: in TwoDimType;
                       OutData: out TwoDimType) is
   begin   -- PASSTHRU
      for row in 1..N loop
```

```
        for col in 1..N loop
           OutData(row,col) := InData(row,col);
        end loop;    -- for col loop
     end loop;    -- for row lopp
  end PASSTHRU;

begin    -- HARNESS
   put_line("% Processed by Pass Thru Module.");
   while (RunFlag) loop
      Header := (others => ' ');
      get_line(Header, StrCount);
      if (Header(Header'first..StrCount) = Dim) then
         get(N);
         put_line(Dim);
         put(N); new_line;
      elsif (Header(Header'first..StrCount) = Pixels) then
         for row in 1..N loop
            for col in 1..N loop
               get(InData(row,col));
            end loop;    -- for col loop
         end loop;    -- for row loop

         PASSTHRU(N, InData, OutData);
         put_line(Pixels);
         for row in 1..N loop
            for col in 1..N loop
               put(OutData(row,col));
            end loop;    -- for col loop
            new_line;
         end loop;    -- for row loop
      elsif(Header(Header'first..StrCount) = Done) then
         put_line(Done);
         RunFlag := FALSE;
      else
         put_line(Header);
      end if;
   end loop;    -- while loop
end HARNESS;
```

3 NON–UNIFORMITY COMPENSATION

3.1 Description

The non–uniformity compensation algorithm provides a pixel–by–pixel correction of the actual pixel response to the desired response. The current algorithm uses up to a five–point, piece-wise–linear correction to the pixel intensity. The correction is determined by sending a specified number of calibration frames through the process. Each of the calibration frames will have been generated by exposing the focal–plane array to a known intensity so that a desired pixel intensity is expected at each pixel.

Dead, or inadequately responsive, pixels are assumed to have been marked by another module and, during processing, they are replaced by the intensity of the previous pixel.

After calibration is performed, the algorithm enters the processing phase. For each pixel which is to be processed, it is first determined if it is a dead pixel. For normal pixels, the calibration intensities are searched to determine which section should be used for correction. After the section is determined, a linear interpolation is performed using the input pixel intensity to interpolate between the desired responses.

3.2 Data Fields

Action	Field Header	Field Data	Data Type
input	Dimensions	FPA dimension	Integer
input	Calibration Frames	Count of calibration frames	Integer
input	Calibration Input	Vector of reference inputs	Integer [1..Count]
input	Calibration Pixel Data	Array of pixel response data for one input reference	Integer [1..Dimension] [1..Dimension]
modified	Pixel Data	Pixel data array	Integer [1..Dimension] [1..Dimension]

3.3 Module Listing

```
--    Program     : NUNICOMP.ada
--                  Non-Uniform Compensation Test Module.
--    Original    : Occam Code -> Fortran Code (MicroSoft)
--    Author      : Rani Indaheng
--                  A. M. Henshaw (Harness)
```

```
--                   Steve Gieseking
--                   Roy W. Melton
--    Date written: Feb 1, 1990
--    Note        : Conforms to the Ga. Tech Algorithm Evaluation
--                   data Format.
---------------------------------------------------------------------
--    Contact     : Rani Indaheng        (404)894-3802
--                   Andrew Henshaw       (404)894-2521
--    Where       : Computer Engineering Research Laboratory
--                   Georgia Institute of Technology
--                   400 Tenth St. CRB 390
--                   Atlanta, GA 30332-0540
---------------------------------------------------------------------
--    Update log :
--    When        Who              What
--    02.06.91    Rani Indaheng    Convert from Fortran to Ada
--    02.13.91    Rani Indaheng    Tested
---------------------------------------------------------------------
with Text_io; use Text_io;
procedure NUNICOMP is
   package Int_io is new Integer_io(integer); use Int_io;
   type OneDimType is array (integer range <>) of integer;
   type TwoDimType is array (integer range <>, integer range <>) of
        integer;
   type ThreeDimType is
        array (integer range <>, integer range <>, integer range <>)
        of integer;

   MaxSize      : constant integer := 128; -- Max FPA size
   MaxCalFrames: constant integer := 5;    -- Default number of frames

   subtype MaxSizeRange is positive range 1..MaxSize;
   subtype MaxCalFramesRange is positive range 1..MaxCalFrames;
   subtype MaxStrSize is positive range 1..72;
   subtype StrType is string(MaxStrSize);

   -- Valid section headers
   Dim       : constant string := "Dimensions";
   Pixels    : constant string := "Pixel Data";
```

```
Done     : constant string := "End";
Sect     : constant string := "Calibration Frames";
CalInput : constant string := "Calibration Input";
CalOutput: constant string := "Calibration Output";

RunFlag: boolean := TRUE;
Header: StrType;
N, StrCount, Count, Sections: integer;
InData, OutData: TwoDimType(MaxSizeRange, MaxSizeRange);
Ic: OneDimType(MaxCalFramesRange);
Oc: ThreeDimType(MaxCalFramesRange, MaxSizeRange, MaxSizeRange);

procedure NON_UNIFORMITY_COMPENSATION(N, Sections: in integer;
                                    Ic: in OneDimType;
                                    Oc: in ThreeDimType;
                                    InData: in TwoDimType;
                                    OutData: in out TwoDimType) is
   Limit: constant integer := 65535;
   LastPixel: integer := 0;
   TmpSection: integer;
begin   -- NON_UNIFORMITY_COMPENSATION
   for row in 1..N loop
      for col in 1..N loop
         if (Oc(1,row,col) = Limit) then
            OutData(row,col) := LastPixel;
         else
            TmpSection := 1;
            while ((TmpSection < (Sections - 1)) and
                  (InData(row,col) >=
                   Oc(TmpSection + 1,row,col))) loop
               TmpSection := TmpSection + 1;
            end loop;   -- while loop
            if (InData(row,col) < Oc(TmpSection,row,col)) then
               OutData(row,col) := Oc(TmpSection,row,col);
            else
               OutData(row,col) :=
                  (InData(row,col) - Oc(TmpSection,row,col))
                  * (Ic(TmpSection + 1) - Ic(TmpSection))
                  / (Oc((TmpSection + 1),row,col)
```

```
                      - Oc(TmpSection,row,col))
                    + Ic(TmpSection);
                end if;
                if (OutData(row,col) > Limit) then
                  OutData(row,col) := Limit;
                end if;

                LastPixel := OutData(row,col);
              end if;
            end loop;    -- for col loop
          end loop;    -- for row loop
        end NON_UNIFORMITY_COMPENSATION;

    begin    -- NUNICOMP
        Count := 1;
        Sections := MaxCalFrames;
        put_line("% Processed by Non-Uniformity Compensation Module.");

        while(RunFlag) loop
          Header := (others => ' ');
          get_line(Header, StrCount);
          if (Header(Header'first..StrCount) = Dim) then
            get(N);
            skip_line;
            put_line(Dim);
            put(N); new_line;
          elsif (Header(Header'first..StrCount) = Sect) then
            get(Sections);
            skip_line;
            put_line(Sect);
            put(Sections); new_line;
          elsif (Header(Header'first..StrCount) = CalInput) then
            if (Count <= Sections) then
              get(Ic(Count));
              skip_line;
              put_line(CalInput);
              put(Ic(Count)); new_line;
            else
```

```
         put_line(CalInput);
      end if;
   elsif (Header(Header'first..StrCount) = CalOutput) then
      if (Count <= Sections) then
         for row in 1..N loop
            for col in 1..N loop
               get(Oc(Count,row,col));
            end loop;   -- for col loop
         end loop;   -- for row loop
         skip_line;

         put_line(CalOutput);
         for row in 1..N loop
            for col in 1..N loop
               put(Oc(Count,row,col));
            end loop;   -- for col loop
            new_line;
         end loop;   -- for row loop
         Count := Count + 1;
      else
         put_line(CalOutput);
      end if;
   elsif (Header(Header'first..StrCount) = Pixels) then
      if ((Count > 1) and (Sections > 1)) then
         for row in 1..N loop
            for col in 1..N loop
               get(InData(row,col));
            end loop;   -- for col loop
         end loop;   -- for row loop
         skip_line;
         if (Count <= Sections) then
            Sections := Count - 1;
         end if;
         NON_UNIFORMITY_COMPENSATION(N, Sections, Ic, Oc, InData,
                                     OutData);
         put_line(Pixels);
         for row in 1..N loop
            for col in 1..N loop
               put(OutData(row,col));
```

```
            end loop;    -- for col loop
            new_line;
         end loop;    -- for row loop
      else
         put_line(Pixels);
      end if;
   elsif (Header(Header'first..StrCount) = Done) then
      put_line(Done);
      RunFlag := FALSE;
   else
      put("NOT = ANY.....");
      put_line(Header);
   end if;
end loop;    -- while loop
end NUNICOMP;
```

4 SPATIAL FILTERING

4.1 Description

The spatial filtering algorithm performs a convolution of the image with a 3x3 coefficient mask. This implementation supports a four point symmetric mask. Separate masks are used for the edge pixels since not all of the pixels which are needed are defined. This allows more general application of boundary conditions than would be available if the undefined pixels were treated as zeros and the same mask was used.

Since the filter allows negative coefficients in the mask, it is possible to generate negative output intensities. The coding allows the intensity to be output limited to a positive range.

4.2 Data Fields

Action	Field Header	Field Data	Data Type
input	Dimensions	FPA dimension	Integer
input	Spatial Filter Control	Filter coefficients (Corner coefficients)	Integer [1..4]
		Filter coefficients (Top coefficients)	Integer [1..4]
		Filter coefficients (Right coefficients)	Integer [1..4]
		Filter coefficients (Center coefficients)	Integer [1..4]
modify	Pixel Data	Pixel data array	Integer [1..Dimension] [1..Dimension]

4.3 Module Listing

```
--   Program      : SPFILT.ada
--                  Spatial Filtering Test Module.
--   Original     : Occam Code -> Fortran Code (MicroSoft)
--   Author       : Rani Indaheng
--                  A. M. Henshaw (Harness)
--                  Steve Gieseking
--                  Roy W. Melton
--   Date written: Jan 23, 1990
--   Note         : Conforms to the Ga. Tech Algorithm Evaluation data
--                  Format.
----------------------------------------------------------------------
--   Contact      : Rani Indaheng        (404)894-3802
--                  Andrew Henshaw       (404)894-2521
--   Where        : Computer Engineering Research Laboratory
--                  Georgia Institute of Technology
--                  400 Tenth St. CRB 390
--                  Atlanta, GA 30332-0540
----------------------------------------------------------------------
--   Update log :
--   When      Who            What
--   02.06.91  Rani Indaheng  Convert from Fortran to Ada
```

```
--   02.13.91  Rani Indaheng  Tested
----------------------------------------------------------------------
with Text_io; use Text_io;
procedure SPFILT is
   package Int_io is new Integer_io(integer); use Int_io;
   type TwoDimType is array (integer range <>, integer range <>) of
        integer;

   MaxSize    : constant integer := 64;  -- Max FPA size
   SFCtrlSize: constant integer := 4;    -- Spatial Control Size

   subtype MaxSizeRange is positive range 1..MaxSize;
   subtype SFCtrlSizeRange is positive range 1..SFCtrlSize;
   subtype MaxStrSize is positive range 1..72;
   subtype StrType is string(MaxStrSize);

   -- Valid section headers
   Dim      : constant string := "Dimensions";
   Pixels   : constant string := "Pixel Data";
   Done     : constant string := "End";
   Controls: constant string := "Spatial Filter Controls";

   RunFlag: boolean := TRUE;
   Header: StrType;
   N, StrCount: integer;
   InData, OutData: TwoDimType(MaxSizeRange, MaxSizeRange);
   C: TwoDimType(SFCtrlSizeRange, SFCtrlSizeRange);

   procedure DEFAULT_FILTER_CTRLS(C: out TwoDimType) is
   begin    -- DEFAULT_FILTER_CTRLS
      for row in 1..4 loop
         for col in 1..3 loop
            C(row,col) := 0;
         end loop;    -- col

         C(row,4) := 16384;
      end loop;    -- row
   end DEFAULT_FILTER_CTRLS;
```

```
procedure SPATIAL_FILTER(N: in integer;
                         C, InData: in TwoDimType;
                         OutData: out TwoDimType) is
LowLimit: constant integer := 0;
UpLimit : constant integer := 65535;
begin   -- SPATIAL_FILTER
   for row in 1..N loop
      for col in 1..N loop
         if (row = 1) then
            if (col = 1) then
               OutData(row,col) := (InData(row+1,col+1) * C(1,1)) +
                                   (InData(row,col+1)   * C(1,2)) +
                                   (InData(row+1,col)   * C(1,3)) +
                                   (InData(row,col)     * C(1,4));
            elsif (col = N) then
               OutData(row,col) := (InData(row+1,col-1) * C(1,1)) +
                                   (InData(row,col-1)   * C(1,2)) +
                                   (InData(row+1,col)   * C(1,3)) +
                                   (InData(row,col)     * C(1,4));
            else
               OutData(row,col) :=
                    ((InData(row+1,col-1) + InData(row+1,col+1)) *
                     C(2,1)) +
                    ((InData(row,col-1)   + InData(row,col+1))   *
                     C(2,2)) +
                    (InData(row+1,col)                           *
                     C(2,3)) +
                    (InData(row,col)                             *
                     C(2,4));
            end if;   -- col
         elsif (row = N) then
            if (col = 1) then
               OutData(row,col) := (InData(row-1,col+1) * C(1,1)) +
                                   (InData(row,col+1)   * C(1,2)) +
                                   (InData(row-1,col)   * C(1,3)) +
                                   (InData(row,col)     * C(1,4));
            elsif (col = N) then
               OutData(row,col) := (InData(row-1,col-1) * C(1,1)) +
                                   (InData(row,col-1)   * C(1,2)) +
```

```
                                (InData(row-1,col)    * C(1,3)) +
                                (InData(row,col)      * C(1,4));
   else
      OutData(row,col) :=
          ((InData(row-1,col-1) + InData(row-1,col+1)) *
           C(2,1)) +
          ((InData(row,col-1)   + InData(row,col+1))   *
           C(2,2)) +
          (InData(row-1,col)                           *
           C(2,3)) +
          (InData(row,col)                             *
           C(2,4));
   end if;   -- col
elsif (col = 1) then
      OutData(row,col) :=
          ((InData(row-1,col+1) + InData(row+1,col+1)) *
           C(3,1)) +
          (InData(row,col+1)                           *
           C(3,2)) +
          ((InData(row-1,col)   + InData(row+1,col))   *
           C(3,3)) +
          (InData(row,col)                             *
           C(3,4));
elsif (col = N) then
   OutData(row,col) :=
          ((InData(row-1,col-1) + Indata(row+1,col-1)) *
           C(3,1)) +
          (InData(row,col-1)                           *
           C(3,2)) +
          ((InData(row-1,col)   + InData(row+1,col))   *
           C(3,3)) +
          (InData(row,col)                             *
           C(3,4));
else
   OutData(row,col) :=
          ((InData(row-1,col-1) + InData(row+1,col-1) +
           InData(row-1,col+1) + InData(row+1,Col+1)) *
           C(4,1)) +
          ((InData(row,col-1)   + InData(row,col+1))   *
```

```
                         C(4,2)) +
                        ((InData(row-1,col)    + InData(row+1,col))   *
                         C(4,3)) +
                        (InData(row,col)                              *
                         C(4,4));
               end if;   -- row
               OutData(row,col) := OutData(row,col) / 16384;
               if (OutData(row,col) < LowLimit) then
                  OutData(row,col) := LowLimit;
               elsif (OutData(row,col) > UpLimit) then
                  OutData(row,col) := UpLimit;
               end if;
            end loop;   -- col
         end loop;   -- row
      end SPATIAL_FILTER;

   begin   -- SPFILT
      put_line("% Processed by Spatial Filtering Module.");
      DEFAULT_FILTER_CTRLS(C);

      while(RunFlag) loop
         Header := (others => ' ');
         get_line(Header, StrCount);
         if (Header(Header'first..StrCount) = Dim) then
            get(N);
            put_line(Dim);
            put(N); new_line;
         elsif (Header(Header'first..StrCount) = Pixels) then
            for row in 1..N loop
               for col in 1..N loop
                  get(InData(row,col));
               end loop;   -- for col loop
            end loop;   -- for row loop
            SPATIAL_FILTER(N, C, InData, OutData);
            put_line(Pixels);
            for row in 1..N loop
               for col in 1..N loop
                  put(OutData(row,col));
               end loop;   -- for col loop
```

```
            new_line;
         end loop;    -- for row loop
      elsif (Header(Header'first..StrCount) = Controls) then
         for row in SFCtrlSizeRange loop
            for col in SFCtrlSizeRange loop
               get(C(row,col));
            end loop;    -- for col loop
         end loop;    -- for row loop
         put_line(Controls);
         for row in SFCtrlSizeRange loop
            for col in SFCtrlSizeRange loop
               put(C(row,col));
            end loop;    -- for col loop
            new_line;
         end loop;    -- for row loop
      elsif (Header(Header'first..StrCount) = Done) then
         put_line(Done);
         RunFlag := FALSE;
      else
         put_line(Header);
      end if;
   end loop;    -- while loop
end SPFILT;
```

5 TEMPORAL FILTERING

5.1 Description

The temporal filtering algorithm provides a pixel–by–pixel infinite impulse response (IIR) filtering of the sequence of images which are sent through the process. This implementation allows up to a fourth–order filter.

5.2 Data Fields

Action	Field header	Field Data	Data Type
input	Dimensions	FPA dimension	Integer
input	Temporal Filtering Limits	Lower limits	Integer
		Upper limits	Integer
modify	Pixel Data	Pixel data array	Integer [1..Dimension] [1..Dimension]

5.3 Module Listing

```
--   Program      : TMPFILT.ada
--                  Temporal Filter Test Module.
--   Original     : Occam Code -> Fortran Code (MicroSoft)
--   Author       : Rani Indaheng
--                  A. M. Henshaw (Harness)
--                  Steve Gieseking
--                  Roy W. Melton
--   Date written: Jan 23, 1990
--   Note         : Conforms to the Ga. Tech Algorithm Evaluation data
--                  Format.
---------------------------------------------------------------------
--   Contact      : Rani Indaheng       (404)894-3802
--                  Andrew Henshaw      (404)894-2521
--   Where        : Computer Engineering Research Laboratory
--                  Georgia Institute of Technology
--                  400 Tenth St. CRB 390
--                  Atlanta, GA 30332-0540
---------------------------------------------------------------------
--   Update log :
--   When      Who            What
--   02.06.91  Rani Indaheng  Convert from Fortran to Ada
---------------------------------------------------------------------
with Text_io; use Text_io;
procedure TMPFILT is
   package Int_io is new Integer_io(integer); use Int_io;
   type OneDimType is array (integer range <>) of integer;
   type TwoDimType is array (integer range <>, integer range <>) of
        integer;

   type FourDimType is array (integer range <>, integer range <>,
        integer range <>, integer range <>) of integer;

   MaxSize   : constant integer := 64;   -- Max FPA size
   TFCtrlSize: constant integer := 24;   -- Temporal Filter size

   subtype MaxSizeRange is positive range 1..MaxSize;
   subtype TFCtrlSizeRange is positive range 1..TFCtrlSize;
```

```
subtype MaxStrSize is positive range 1..72;
subtype StrType is string(MaxStrSize);

-- Valid section headers
Dim   : constant string := "Dimensions";
Pixels: constant string := "Pixel Data";
Done  : constant string := "End";
Limits: constant string := "Temporal Filtering Limits";

RunFlag: boolean := TRUE;
Lower  : integer := 0;       -- default value
Upper  : integer := 32767;    -- default value
Header: StrType;
N, StrCount: integer;
InData, OutData: TwoDimType(MaxSizeRange, MaxSizeRange);
C: OneDimType(TFCtrlSizeRange);

procedure TMP_FILTER(N: in integer;
                    C: in OneDimType;
                    InData: in TwoDimType;
                    OutData: out TwoDimType) is

  TF_A0                 : constant integer := 1;
  TF_A1                 : constant integer := 2;
  TF_A2                 : constant integer := 3;
  TF_B0                 : constant integer := 4;
  TF_B1                 : constant integer := 5;
  TF_B2                 : constant integer := 6;
  TF_SCALE_STATE        : constant integer := 7;
  TF_SCALE_OUTPUT       : constant integer := 8;
  TF_UPPER_LIMIT_STATE  : constant integer := 9;
  TF_LOWER_LIMIT_STATE  : constant integer := 10;
  TF_UPPER_LIMIT_OUTPUT: constant integer := 11;
  TF_LOWER_LIMIT_OUTPUT: constant integer := 12;
   Xnew, Ynew, Ptr, Value: integer;
   X: FourDimType(MaxSizeRange, MaxSizeRange, 1..2, 1..2);

begin   -- TMP_FILTER
   for i in MaxSizeRange loop
```

```
      for j in MaxSizeRange loop
         for k in 1..2 loop
            for l in 1..2 loop
               X(i,j,k,l) := 0;
            end loop;   -- l
         end loop;   -- k
      end loop;   -- j
   end loop;   -- i
   for i in 1..N loop
      for j in 1..N loop
         Value := InData(i,j);
         for k in 1..2 loop
            Ptr := (k - 1) * 12;
            Xnew := ((C(Ptr + TF_A0) * Value) +
                     (C(Ptr + TF_A1) * X(i,j,k,1)) +
                     (C(Ptr + TF_A2) * X(i,j,k,2))) /
                    C(Ptr + TF_SCALE_STATE);
            Ynew := ((C(Ptr + TF_B0) * Value) +
                     (C(Ptr + TF_B1) * X(i,j,k,1)) +
                     (C(Ptr + TF_B2) * X(i,j,k,2))) /
                    C(Ptr + TF_SCALE_OUTPUT);
            X(i,j,k,2) := X(i,j,k,1);

            if (Xnew > C(Ptr+TF_UPPER_LIMIT_STATE)) then
               X(i,j,k,1) := C(Ptr+TF_UPPER_LIMIT_STATE);
            elsif (Xnew < C(Ptr+TF_LOWER_LIMIT_STATE)) then
               X(i,j,k,1) := C(Ptr+TF_LOWER_LIMIT_STATE);
            else
               X(i,j,k,1) := Xnew;
            end if;
            if (Ynew > C(Ptr+TF_UPPER_LIMIT_OUTPUT)) then
               Value := C(Ptr+TF_UPPER_LIMIT_OUTPUT);
            elsif (Ynew < C(Ptr+TF_LOWER_LIMIT_OUTPUT)) then
               Value := C(Ptr+TF_LOWER_LIMIT_OUTPUT);
            else
               Value := Ynew;
            end if;
         end loop;   -- k
```

```
            OutData(i,j) := Value;
         end loop;   -- j
      end loop;   -- i
   end TMP_FILTER;

   procedure CALCULATE_FILTER_CTRLS(Lower, Upper: in integer;
                                    C: out OneDimType) is
      Step: integer := 0;
   begin   -- CALCULATE_FILTER_CTRLS
      for i in 1..2 loop
         for j in 1..8 loop
            C(j+Step) := 1;
         end loop;   -- j
         C(9+Step)  := 3;
         C(10+Step) := 1;
         C(11+Step) := Upper;
         C(12+Step) := Lower;
         Step := Step + 12;
      end loop;   -- i

   end CALCULATE_FILTER_CTRLS;

begin   -- TMPFILT
   put_line("% Processed by Temporal Filtering Module.");
   while(RunFlag) loop
      Header := (others => ' ');
      get_line(Header, StrCount);
      if (Header(Header'first..StrCount) = Dim) then
         get(N);
         put_line(Dim);
         put(N); new_line;
      elsif (Header(Header'first..StrCount) = Pixels) then
         for row in 1..N loop
            for col in 1..N loop
               get(InData(row,col));
            end loop;   -- for col loop
         end loop;   -- for row loop
```

```
         CALCULATE_FILTER_CTRLS(Lower, Upper, C);
         TMP_FILTER(N, C, InData, OutData);
         put_line(Pixels);
         for row in 1..N loop
            for col in 1..N loop
               put(OutData(row,col));
            end loop;   -- for col loop
            new_line;
         end loop;   -- for row loop
      elsif (Header(Header'first..StrCount) = Limits) then
         get(Lower);
         get(Upper);
         put_line(Limits);
         put(Lower);
         put(' ');
         put(Upper); new_line;
      elsif (Header(Header'first..StrCount) = Done) then
         put_line(Done);
         RunFlag := FALSE;
      else
         put_line(Header);
      end if;
   end loop;   -- while loop
end TMPFILT;
```

6 THRESHOLDING

The thresholding algorithm is used to partition the image into points which are of interest and those that are not of interest. Pixels are zeroed if they are not of interest. A pixel is passed if the intensity is above a calculated lower threshold value and below a fixed upper threshold value. The lower threshold supports two of the modes which are in the Georgia Tech VLSI design. This includes a simple, fixed threshold and an adaptive threshold based on the average and first central absolute moment of the surrounding eight pixels.

6.1 SIMPLE THRESHOLDING

6.1.1 Data Fields

Action	Field Header	Field Data	Data Type
input	Dimensions	FPA dimension	Integer
input	Simple Thresholding Limits	Lower limit	Integer
		Upper limit	Integer
output	Simple Thresholding Statistics	Lower limit used	Integer
		Upper limit used	Integer
		Count of pixels exceeding limit	Integer
modify	Pixel Data	Pixel data array	Integer [1..Dimension] [1..Dimension]

6.1.2 Module Listing

```
--    Program      : STHRESH.ada
--                   Simple Thresholding Test Module.
--    Original     : Occam Code -> Fortran Code (MicroSoft)
--    Author       : Rani Indaheng
--                   A. M. Henshaw (Harness)
--                   Steve Gieseking
--                   Roy W. Melton
--    Date written: Jan 23, 1990
--    Note         : Conforms to the Ga. Tech Algorithm Evaluation Data
--                   Format.
---------------------------------------------------------------------
--    Contact      : Rani Indaheng        (404)894-3802
--                   Andrew Henshaw       (404)894-2521
--    Where        : Computer Engineering Research Laboratory
--                   Georgia Institute of Technology
--                   400 Tenth St. CRB 390
--                   Atlanta, GA 30332-0540
---------------------------------------------------------------------
--    UPDATE LOG
--    When       Who            What
--    02.07.91   Rani Indaheng  Convert from Fortran to Ada
```

```
--   02.13.91  Rani Indaheng  Tested
-----------------------------------------------------------------------
with Text_io; use Text_io;
procedure STHRESH is
   package Int_io is new Integer_io(integer); use Int_io;
   type TwoDimType is array (integer range <>, integer range <>) of
        integer;

   MaxSize: constant integer := 128;   -- Max FPA size

   subtype MaxSizeRange is positive range 1..MaxSize;
   subtype MaxStrSize is positive range 1..72;
   subtype StrType is string(MaxStrSize);

   -- Valid section headers
   Dim   : constant string := "Dimensions";
   Pixels: constant string := "Pixel Data";
   Done  : constant string := "End";
   Limits: constant string := "Simple Thresholding Limits";

   Lower  : integer := 0;       -- Default value
   Upper  : integer := 32767;   -- Default value
   RunFlag: boolean := TRUE;
   Header: StrType;
   N, StrCount: integer;
   InData, OutData: TwoDimType(MaxSizeRange, MaxSizeRange);

   procedure SMP_THRSH(N, Lower, Upper: in integer;
                       InData: in TwoDimType;
                       OutData: out TwoDimType) is
      Count: integer := 0;
      Pixel: integer;

   begin   -- SMP_THRSH
      for row in 1..N loop
         for col in 1..N loop
            Pixel := InData(row,col);
            if ((Pixel >= Lower) and (Pixel <= Upper)) then
               Count := Count + 1;
```

```
               OutData(row,col) := Pixel;
            else
               OutData(row,col) := 0;
            end if;
         end loop;   -- col
      end loop;   -- row
      put_line("Simple Thresholding Statistics");
      put(Lower); put(' ');
      put(Upper); put(' ');
      put(Count); new_line;
   end SMP_THRSH;

begin   -- STHRESH
   put_line("% Processed by Simple Thresholding Module.");

   while(RunFlag) loop
      Header := (others => ' ');
      get_line(Header, StrCount);
      if (Header(Header'first..StrCount) = Dim) then
         get(N);
         put_line(Dim);
         put(N); new_line;
      elsif (Header(Header'first..StrCount) = Pixels) then
         for row in 1..N loop
            for col in 1..N loop
               get(InData(row,col));
            end loop;   -- for col loop
         end loop;   -- for row loop
         SMP_THRSH(N, Lower, Upper, InData, OutData);
         put_line(Pixels);
         for row in 1..N loop
            for col in 1..N loop
               put(OutData(row,col)); put("  ");
            end loop;   -- for col loop
            new_line;
         end loop;   -- for row loop
      elsif (Header(Header'first..StrCount) = Limits) then
         get(Lower);
         get(Upper);
```

```
            put_line(Limits);
            put(Lower);
            put(' ');
            put(Upper); new_line;
         elsif (Header(Header'first..StrCount) = Done) then
            put_line(Done);
            RunFlag := FALSE;
         else
            put_line(Header);
         end if;
      end loop;    -- while loop
end STHRESH;
```

6.2 ADAPTIVE THRESHOLDING

6.2.1 Data Fields

Action	Field Header	Field Data	Data Type
input	Dimensions	FPA dimension	Integer
input	Adaptive Thresholding Parameters	Upper limit	Integer
		k1	Integer
		k2	Integer
		k3	Integer
		Scale	Integer
output	Adaptive Thresholding Statistics	Upper limit used	Integer
		Count of pixels exceeding limit	Integer
modify	Pixel Data	Pixel data array	Integer [1..Dimension] [1..Dimension]

6.2.2 Module Listing

```
--    Program       : ADTHRESH.ada
--                    Adaptive Thresholding Test Module.
--    Original      : Occam Code -> Fortran Code (MicroSoft)
--    Author        : Rani Indaheng
--                    A. M. Henshaw (Harness)
```

```
--                  Steve Gieseking
--                  Roy W. Melton
--   Date written: Jan 23, 1990
--   Note         : Conforms to the Ga. Tech Algorithm Evaluation Data
--                  Format.
----------------------------------------------------------------------
--   Contact      : Rani Indaheng       (404)894-3802
--                  Andrew Henshaw      (404)894-2521
--   Where        : Computer Engineering Research Laboratory
--                  Georgia Institute of Technology
--                  400 Tenth St. CRB 390
--                  Atlanta, GA 30332-0540
----------------------------------------------------------------------
--   UPDATE/TEST LOG
--   When      Who          What
--   02.07.91  R. Indaheng  Convert from Fortran to Ada
--   02.13.91  R. Indaheng  Checked with Roy's original
----------------------------------------------------------------------
with Text_io; use Text_io;
procedure ADTHRESH is
   package Int_io is new Integer_io(integer); use Int_io;

   type TwoDimType is array (integer range <>, integer range <>) of
        integer;

   MaxSize: constant integer := 64;    -- Max FPA size

   subtype MaxSizeRange is positive range 1..MaxSize;
   subtype MaxStrSize is positive range 1..72;
   subtype StrType is string(MaxStrSize);

   -- Valid section headers
   Dim    : constant string := "Dimensions";
   Pixels: constant string := "Pixel Data";
   Done   : constant string := "End";
   Parms  : constant string := "Adaptive Thresholding Parameters";

   K1      : integer := 1;
   K2      : integer := 0;
```

```
K3      : integer := 0;
Scale   : integer := 8;
Upper   : integer := 32767;    -- Default values
RunFlag: boolean := TRUE;
Header           : StrType;
N, StrCount      : integer;
InData, OutData: TwoDimType(MaxSizeRange, MaxSizeRange);

procedure ADP_THRSH(N, Upper, K1, K2, K3, Scale: in integer;
                    InData: in TwoDimType;
                    OutData: out TwoDimType) is
   Count: integer := 0;
   Sum: integer := 0;
   Average, Stat, Lower, k, l: integer;

 begin   -- ADP_THRSH
   for i in 1..N loop
      for j in 1..N loop
         if (((i = 1) or (i = N)) OR ((j = 1) or (j = N))) then
            OutData(i,j) := 0;
         else
            Average := InData(i-1,j-1) + InData(i-1,j) +
                       InData(i-1,j+1) + InData(i,j-1) +
                       InData(i,j+1)   + InData(i+1,j-1) +
                       InData(i+1,j)   + InData(i+1,j+1);
            Stat := 0;
            k := -2;
            l := -2;
            for indx_k in 1..3 loop
               k := k + 1;
               for indx_l in 1..3 loop
                  l := l + 1;
                  if ((k /= 0) and (l /= 0)) then
                     Stat := Stat + abs((InData(i+k,j+l) * 8) -
                             Average);
                  end if;
               end loop;   -- indx_l
            end loop;   -- indx_k
```

```
            Lower := ((Average * K1) + (Stat * K2) + K3) / Scale;
            Sum := Sum + Stat;
            if ((InData(i,j) >= Lower) and
                 (InData(i,j) <= Upper)) then
               OutData(i,j) := InData(i,j);
               Count := Count + 1;
            else
               OutData(i,j) := 0;
            end if;
         end if;
      end loop;   -- j
   end loop;   -- i
   put_line("Adaptive Thresholding Statistics");
   put(Upper); put(' ');
   put(Count); new_line;
end ADP_THRSH;

begin   -- ADTHRESH
   put_line("% Processed by Adaptive Thresholding Module.");

   while(RunFlag) loop
      Header := (others => ' ');
      get_line(Header, StrCount);
      if (Header(Header'first..StrCount) = Dim) then
         get(N);
         put_line(Dim);
         put(N); new_line;
      elsif (Header(Header'first..StrCount) = Pixels) then
         for row in 1..N loop
            for col in 1..N loop
               get(InData(row,col));
            end loop;   -- for col loop
         end loop;   -- for row loop
         ADP_THRSH(N, Upper, K1, K2, K3, Scale, InData, OutData);
         put_line(Pixels);
         for row in 1..N loop
            for col in 1..N loop
               put(OutData(row,col));
            end loop;   -- for col loop
```

```
            new_line;
         end loop;    -- for row loop
      elsif (Header(Header'first..StrCount) = Parms) then
         get(Upper);
         get(K1);
         get(K2);
         get(K3);
         get(Scale);
         put_line(Parms);
         put(Upper); put(' ');
         put(K1);    put(' ');
         put(K2);    put(' ');
         put(K3);    put(' ');
         put(Scale); new_line;
      elsif (Header(Header'first..StrCount) = Done) then
         put_line(Done);
         RunFlag := FALSE;
      else
         put_line(Header);
      end if;
   end loop;    -- while loop
end ADTHRESH;
```

7 CLUSTERING AND CENTROIDING

7.1 Description

The clustering algorithm forms connected sets of pixels based on the surrounding pixels. Two pixels are elements of the same cluster of pixels if they are one of the eight nearest neighbors of each other. The centroiding algorithm calculates the area centroid and the intensity weighted centroid of the clusters specified by the clustering algorithm.

7.2 Data Fields

Action	Field Header	Field Data	Data Type
input	Dimensions	FPA dimension	Integer
input	Pixel Data	Pixel data array	Integer [1..Dimension] [1..Dimension]
output	Clusters	Cluster count	Integer
output	Centroids	Vector of the following repeated Cluster count times	
		Area centroid (X)	Integer
		Area centroid (Y)	Integer
		Intensity centroid (X)	Integer
		Intensity Centroid (Y)	Integer
		Area in pixels	Integer
		Total cluster intensity	Integer

7.3 Module Listing

```
--    Program      : CENTROID.ada
--                   Clustering and Centroiding Test Module.
--    Original     : Occam Code -> Fortran Code (MicroSoft)
--    Author       : Rani Indaheng
--                   A. M. Henshaw (Harness)
--                   Steve Gieseking
--                   Roy W. Melton      Feb 12, 1990
--    Date written: Jan 23, 1990
--    Note         : Conforms to the Ga. Tech Algorithm Evaluation Data
--                   Format.
---------------------------------------------------------------------
--    Contact      : Rani Indaheng       (404)894-3802
--                   Andrew Henshaw      (404)894-2521
--    Where        : Computer Engineering Research Laboratory
--                   Georgia Institute of Technology
--                   400 Tenth St. CRB 390
--                   Atlanta, GA 30332-0540
---------------------------------------------------------------------
--    UPDATE LOG
```

```
--   When      Who             What
--   02.07.91  Rani Indaheng   Convert from Fortran to Ada
--   02.13.91  Rani Indaheng   Tested
----------------------------------------------------------------------

with Text_io; use Text_io;
procedure CENTROID is
   package Int_io is new Integer_io(integer); use Int_io;

   type OneDimType is array (integer range <>) of integer;
   type TwoDimType is array (integer range <>, integer range <>) of
        integer;

   MaxSize       : constant integer := 64;      -- Max FPA size
   MaxClusterSize: constant integer := 1024;   -- Cluster size

   subtype MaxSizeRange is positive range 1..MaxSize;
   subtype MaxClusterSizeRange is positive range 1..MaxClusterSize;
   subtype MaxStrSize is positive range 1..72;
   subtype StrType is string(MaxStrSize);

   -- Valid section headers
   Dim   : constant string := "Dimensions";
   Pixels: constant string := "Pixel Data";
   Done  : constant string := "End";

   RunFlag      : boolean := TRUE;
   Header       : StrType;
   N, StrCount  : integer;
   ClusterCount: integer;
   Frame        : TwoDimType(MaxSizeRange, MaxSizeRange);
   Clusters     : TwoDimType(MaxClusterSizeRange, 1..6);

   function SHIFT_RIGTH(Arg: integer) return integer is
   begin   -- SHIFT_RIGTH
      return (Arg / 2);
   end SHIFT_RIGTH;

   procedure CENTROID_IMAGE(N: in integer;
                            Frame: in TwoDimType;
```

```
                               ClusterCount: in out integer;
                               Clusters: in out TwoDimType) is
   Csum      : constant integer := 1;
   CsumX     : constant integer := 2;
   CsumY     : constant integer := 3;
   Isum      : constant integer := 4;
   IsumX     : constant integer := 5;
   IsumY     : constant integer := 6;
   AcoorX    : constant integer := 1;
   AcoorY    : constant integer := 2;
   IcoorX    : constant integer := 3;
   IcoorY    : constant integer := 4;
   Area      : constant integer := 5;
   Intensity: constant integer := 6;
   C0, C1, Cnm1, Cn, Cnp1, FinalCluster: integer;
   Reassign    : OneDimType(0..MaxClusterSize-1);
   TmpClusters: OneDimType(1..MaxSize+1);
   Tmp         : OneDimType(1..6);

begin   -- CENTROID_IMAGE
   for i in 1..N+1 loop
      TmpClusters(i) := 0;
   end loop;    -- i
   FinalCluster := 0;
   Reassign(0) := 0;
   for i in 1..N loop
      -- Initialize row
      C1 := 0;
      Cnp1 := 0;
      Cn := TmpClusters(1);
      while (Cn /= Reassign(Cn)) loop
         Cn := Reassign(Cn);
      end loop;    -- while
      for j in 1..N loop
         Cnm1 := TmpClusters(j+1);
         while (Cnm1 /= Reassign(Cnm1)) loop
            Cnm1 := Reassign(Cnm1);
         end loop;    -- while
```

```
if (Frame(i,j) = 0) then
   C0 := 0;
elsif (C1 /= 0) then
   -- Add pixel to cluster
   if ((Cnm1 /= 0) and (Cnm1 /= C1)) then
      -- Merge C1, Cnm1
      Clusters(C1,Csum)   := Clusters(C1,Csum) +
                             Clusters(Cnm1,Csum) + 1;
      Clusters(C1,CsumX) := Clusters(C1,CsumX) +
                             Clusters(Cnm1,CsumX) + j;
      Clusters(C1,CsumY) := Clusters(C1,CsumY) +
                             Clusters(Cnm1,CsumY) + i;
      Clusters(C1,Isum)   := Clusters(C1,Isum) +
                             Clusters(Cnm1,Isum) +
                              Frame(i,j);
      Clusters(C1,IsumX) := Clusters(C1,IsumX) +
                             Clusters(Cnm1,IsumX) +
                             (Frame(i,j) * j);
      Clusters(C1,IsumY) := Clusters(C1,IsumY) +
                             Clusters(Cnm1,CsumY) +
                             (Frame(i,j) * i);
      Clusters(Cnm1,Csum) := 0;
      Reassign(Cnm1) := C1;
      Cnm1 := C1;
      Cn := Reassign(Cn);
      C0 := C1;
   else
      -- Add to C1
      Clusters(C1,Csum)   := Clusters(C1,Csum) + 1;
      Clusters(C1,CsumX) := Clusters(C1,CsumX) + j;
      Clusters(C1,CsumY) := Clusters(C1,CsumY) + i;
      Clusters(C1,Isum)   := Clusters(C1,Isum) +
                              Frame(i,j);
      Clusters(C1,IsumX) := Clusters(C1,IsumX) +
                             (Frame(i,j) * j);
      Clusters(C1,IsumY) := Clusters(C1,IsumY) +
                             (Frame(i,j) * i);
      C0 := C1;
   end if;
```

```
      elsif (Cn /= 0) then
         -- Add to Cn
         Clusters(Cn, Csum)   := Clusters(Cn, Csum) + 1;
         Clusters(Cn, CsumX)  := Clusters(Cn, CsumX) + j;
         Clusters(Cn, CsumY)  := Clusters(Cn, CsumY) + i;
         Clusters(Cn, Isum)   := Clusters(Cn, Isum) + Frame(i,j);
         Clusters(Cn, IsumX)  := Clusters(Cn, IsumX) +
                                 (Frame(i,j) * j);
         Clusters(Cn, IsumY)  := Clusters(Cn, IsumY) +
                                 (Frame(i,j) * i);
         C0 := Cn;
      elsif (Cnm1 /= 0) then
         if ((Cnp1 /= 0) and (Cnp1 /= Cnm1)) then
            -- Merge Cnm1, Cnp1
            Clusters(Cnm1,Csum)   := Clusters(Cnm1,Csum) +
                                     Clusters(Cnp1,Csum) + 1;
            Clusters(Cnm1,CsumX)  := Clusters(Cnm1,CsumX) +
                                     Clusters(Cnp1,CsumX) + j;
            Clusters(Cnm1,CsumY)  := Clusters(Cnm1,CsumY) +
                                     Clusters(Cnp1,CsumY) + i;
            Clusters(Cnm1,Isum)   := Clusters(Cnm1,Isum) +
                                     Clusters(Cnp1,Isum) +
                                      Frame(i,j);
            Clusters(Cnm1,IsumX)  := Clusters(Cnm1,IsumX) +
                                     Clusters(Cnp1,IsumX) +
                                     (Frame(i,j) * j);
            Clusters(Cnm1,IsumY)  := Clusters(Cnm1,IsumY) +
                                     Clusters(Cnp1,CsumY) +
                                     (Frame(i,j) * i);
            Clusters(Cnp1,Csum) := 0;
            Reassign(Cnp1) := Cnm1;
            C0 := Cnm1;
         else
            -- Add to Cnm1
            Clusters(Cnm1, Csum)   := Clusters(Cnm1, Csum) + 1;
            Clusters(Cnm1, CsumX)  := Clusters(Cnm1, CsumX) + j;
            Clusters(Cnm1, CsumY)  := Clusters(Cnm1, CsumY) + i;
            Clusters(Cnm1, Isum)   := Clusters(Cnm1, Isum) +
                                      Frame(i,j);
```

```
                Clusters(Cnml, IsumX) := Clusters(Cnml, IsumX) +
                                         (Frame(i,j) * j);
                Clusters(Cnml, IsumY) := Clusters(Cnml, IsumY) +
                                         (Frame(i,j) * i);
                C0 := Cnml;
             end if;
          elsif (Cnpl /= 0) then
             -- Add to Cnpl
             Clusters(Cnpl, Csum)  := Clusters(Cnpl, Csum) + 1;
             Clusters(Cnpl, CsumX) := Clusters(Cnpl, CsumX) + j;
             Clusters(Cnpl, CsumY) := Clusters(Cnpl, CsumY) + i;
             Clusters(Cnpl, Isum)  := Clusters(Cnpl, Isum) +
                                      Frame(i,j);
             Clusters(Cnpl, IsumX) := Clusters(Cnpl, IsumX) +
                                      (Frame(i,j) * j);
             Clusters(Cnpl, IsumY) := Clusters(Cnpl, IsumY) +
                                      (Frame(i,j) * i);
             C0 := Cnpl;

          else
             -- New Cluster
             FinalCluster := FinalCluster + 1;
             C0 := FinalCluster;
             Clusters(C0,Csum)   := 1;
             Clusters(C0,CsumX)  := j;
             Clusters(C0,CsumY)  := i;
             Clusters(C0,Isum)   := Frame(i,j);
             Clusters(C0,IsumX)  := (Frame(i,j) * j);
             Clusters(C0,IsumY)  := (Frame(i,j) * i);
             Reassign(C0) := C0;
          end if;
          TmpClusters(j) := C0;
          -- Update for next column
          C1 := C0;
          Cnpl := Cn;
          Cn := Cnml;
       end loop;   -- j
    end loop;   -- i
```

```
      -- Output Centroids
      ClusterCount := 0;
      for i in 1..FinalCluster loop
         if (Clusters(i,Csum) /= 0) then
            -- Valid cluster
            Tmp(AcoorX) := (Clusters(i,CsumX) +
                           SHIFT_RIGTH(Clusters(i,Csum))) /
                           Clusters(i, Csum);
            Tmp(AcoorY) := (Clusters(i,CsumY) +
                           SHIFT_RIGTH(Clusters(i,Csum))) /
                           Clusters(i, Csum);
            Tmp(IcoorX) := (Clusters(i,IsumX) +
                           SHIFT_RIGTH(Clusters(i,Isum))) /
                           Clusters(i, Isum);
            Tmp(IcoorY) := (Clusters(i,IsumY) +
                           SHIFT_RIGTH(Clusters(i,Isum))) /
                           Clusters(i, Isum);
            Tmp(Area)   := Clusters(i,Csum);
            Tmp(Intensity) := Clusters(i,Isum);
            ClusterCount := ClusterCount + 1;
            for j in 1..6 loop
               Clusters(ClusterCount,j) := Tmp(j);
            end loop;   -- j
         end if;
      end loop;   -- i
   end CENTROID_IMAGE;

begin   -- CENTROID
   put_line("% Processed by Centroid Image Module.");

   while(RunFlag) loop
      Header := (others => ' ');
      get_line(Header, StrCount);
      if (Header(Header'first..StrCount) = Dim) then
         get(N);
         put_line(Dim);
         put(N); new_line;
      elsif (Header(Header'first..StrCount) = Pixels) then
         for row in 1..N loop
```

```
            for col in 1..N loop
               get(Frame(row,col));
            end loop;    -- for col loop
         end loop;    -- for row loop
         CENTROID_IMAGE(N, Frame, ClusterCount, Clusters);
         put_line(Pixels);
         for row in 1..N loop
            for col in 1..N loop
               put(Frame(row,col));
            end loop;    -- for col loop
            new_line;
         end loop;    -- for row loop

         put_line("Clusters");
         put(ClusterCount); new_line;

         if (ClusterCount > 0) then
            put_line("Centroids");
            for row in 1..ClusterCount loop
               for col in 1..6 loop
                  put(Clusters(row,col));
               end loop;    -- col
               new_line;
            end loop;     -- row
         end if;
      elsif (Header(Header'first..StrCount) = Done) then
         put_line(Done);
         RunFlag := FALSE;
      else
         put_line(Header);
      end if;
   end loop;    -- while loop
end CENTROID;
```